Discrete Applied Mathematics 158 (2010) 1879–1887

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Reoptimizing the 0–1 knapsack problem

Claudia Archetti, Luca Bertazzi *, M. Grazia Speranza

University of Brescia, Department of Quantitative Methods, Brescia, Italy

ARTICLE INFO

Article history:
Received 27 August 2009
Received in revised form 8 July 2010
Accepted 3 August 2010
Available online 21 August 2010

Keywords:
0–1 knapsack problem
Reoptimization
Worst-case analysis

ABSTRACT

In this paper we study the problem where an optimal solution of a knapsack problem on n items is known and a very small number k of new items arrive. The objective is to find an optimal solution of the knapsack problem with $n + k$ items, given an optimal solution on the n items (reoptimization of the knapsack problem). We show that this problem, even in the case $k = 1$, is NP-hard and that, in order to have effective heuristics, it is necessary to consider not only the items included in the previously optimal solution and the new items, but also the discarded items. Then, we design a general algorithm that makes use, for the solution of a subproblem, of an α-approximation algorithm known for the knapsack problem. We prove that this algorithm has a worst-case performance bound of $\frac{1}{2-\alpha}$, which is always greater than α, and therefore that this algorithm always outperforms the corresponding α-approximation algorithm applied from scratch on the $n+k$ items. We show that this bound is tight when the classical *Ext-Greedy* algorithm and the $G\frac{3}{4}$ algorithm are used to solve the subproblem. We also show that there exist classes of instances on which the running time of the reoptimization algorithm is smaller than the running time of an equivalent PTAS and FPTAS.

0. Introduction

Combinatorial problems have been widely studied under the assumption that the instance is completely known at the time the optimization is carried out, that no information is available on the solution of the instance or of related instances and that an optimal solution will not need to be modified later. However, it happens more and more frequently that new information is made available in real time, after an optimal solution has been obtained. Therefore, the instance is perturbed and the optimal solution needs to be modified accordingly, in a short amount of time. A typical approach to solve this problem is to apply a known heuristic algorithm from scratch, completely ignoring the work done to obtain the previously optimal solution. A different approach is to solve the so-called *reoptimization problem*, that is the problem of finding an optimal solution to the perturbed instance knowing an optimal solution to the original one. Two issues become of interest in this setting. The first is the complexity of the reoptimization problem. The second is to study whether it is possible to take advantage of the available optimal solution to design more effective and/or efficient heuristic algorithms. This approach has received little attention in the past. It was applied to a scheduling problem by Schäffter [13] and to the traveling salesman problem by Archetti et al. [1] and Ausiello et al. [2]. Bockenhauer et al. [3] investigated the hardness of solving a modified instance of a problem when an optimal solution of the original instance is known for some variants of the traveling salesman problem and the Steiner tree problem.

The 0–1 knapsack problem is one of the most studied combinatorial optimization problems. Very effective exact algorithms are known as well as several heuristics, approximation algorithms and schemes (we refer to the excellent books by Martello and Toth [11] and by Kellerer et al. [9]). In particular, several Polynomial Time Approximation Schemes (PTASs)

* Corresponding author. Fax: +39 0302400925.
E-mail addresses: archetti@eco.unibs.it (C. Archetti), bertazzi@eco.unibs.it (L. Bertazzi), speranza@eco.unibs.it (M.G. Speranza).

0166-218X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2010.08.003

and Fully Polynomial Time Approximation Schemes (FPTASs) are known. Any PTAS for the knapsack problem is based on the idea of guessing a set of items included in an optimal solution by going through all possible candidate sets and then filling the remaining capacity by applying a greedy algorithm. We recall the classical PTAS by Sahni [12] and the PTAS by Caprara et al. [4], which improves the PTAS by Sahni [12] in terms of running time. Any FPTAS is based on the idea of scaling the profit values and then applying dynamic programming on the resulting instance. We recall the earliest FPTAS by Ibarra and Kim [6], the classical FPTAS by Lawler [10] and the best known FPTAS by Kellerer and Pferschy [7,9].

In this paper we study the reoptimization of the 0–1 knapsack problem. The situation we intend to study is the following. An exact algorithm is available for the solution of large size instances and an optimal solution to an instance has been found. This may have required a large amount of time. Then, a small perturbation of the optimally solved instance takes place. A very small number k of new items become known and available. In this type of setting, the reoptimization problem becomes very interesting. Some of the new items might be selected together with or instead of some of the items included in the previously optimal solution. Moreover, some of the previously discarded items might become interesting if combined with some of the new ones. Little time is available to react to the perturbation of the instance and improve the previously optimal solution, and a simple to implement and run heuristic is desired to achieve this goal.

The purpose of this paper is to understand the nature of the reoptimization of a classical problem such as the 0–1 knapsack and answer a number of natural questions in this setting. Given an optimal solution on n items, is the problem on $n + k$ items hard? How good is a solution that ignores the previously discarded items? Can we design a simple to implement and run heuristic with guaranteed performance that takes advantage of the previously optimal solution? We will show that the reoptimization of the knapsack problem remains an NP-hard problem. This means that, from the complexity point of view, the availability of the previously optimal solution is not helpful. We will also show that, in order to have effective heuristics, it is necessary to consider not only the items included in the previously optimal solution and the new items, but also the discarded items. Then, we will present a general algorithm that makes use, for the solution of a subproblem, of an α-approximation algorithm known for the knapsack problem, where α is the worst-case ratio between the value of the solution of the approximation algorithm and the value of an optimal solution. This algorithm has a worst-case performance bound of $\frac{1}{2-\alpha}$, which is always greater than α, and therefore this algorithm always outperforms the corresponding α-approximation algorithm applied from scratch on the perturbed instance. If a simple α-approximation algorithm is used, the proposed heuristic is simple to implement and run. We prove that this bound is tight when the simple and classical *Ext-Greedy* algorithm and the simple $G^{\frac{3}{4}}$ algorithm are used to solve the subproblem. We also show that there exist classes of instances on which the running time of the reoptimization algorithm is smaller than the running time of an equivalent PTAS and FPTAS.

The paper is organized as follows. In Section 1 the reoptimization problem is described. In Section 2 we show that this problem is NP-hard. In Section 3 we prove that heuristic algorithms that find the new solution on the basis of the new items and the items selected in the previously optimal solution only are not effective. Finally, in Section 4 we describe the general $\frac{1}{2-\alpha}$-approximation algorithm, show the tightness of this algorithm when the *Ext-Greedy* and the $G^{\frac{3}{4}}$ algorithms are used to solve the subproblem and compare its performance with the best PTAS and FPTAS.

1. Problem description

In the classical knapsack problem, referred to as Problem K_n, a knapsack of capacity b and a set U_n of n items are available. Each item $u \in U_n$ has a positive weight $w(u)$, with $w(u) \leq b$, and a positive profit $p(u)$. The aim is to find a subset of U_n that maximizes the total profit and satisfies the capacity b of the knapsack. We denote by U_n^* and z_n^* an optimal subset and the corresponding profit, respectively. A heuristic solution of this problem can be found by applying a greedy algorithm (see [11,9]). The solution given by this algorithm is the best between the following two solutions. The first is obtained by first ordering the items in the non-increasing order of $p(u)/w(u)$ and then, following this ordering, inserting them into the knapsack. The second is obtained by inserting into the knapsack the item with maximum profit only. This algorithm has a tight worst-case performance bound of 1/2. We will refer to it as *Ext-Greedy*, as in [9], to differentiate it from the basic greedy algorithm that does not consider the item with maximum profit. A better approximation algorithm can be obtained by considering a pair of items, inserting them into the knapsack and then applying the *Ext-Greedy* algorithm to a subset of the remaining items on the residual capacity. The subset of the remaining items is formed by only those items with profit not greater than the minimum profit of the two items in the initial pair. The algorithm considers each pair of items and applies the previous procedure choosing the solution that maximizes the total profit. This algorithm is referred to as $G^{\frac{3}{4}}$ and has a tight worst-case performance bound of 3/4 (see [9]).

The problem we study, referred to as Problem RK_{n+k}, is the problem in which, given an optimal solution U_n^* of the Problem K_n, a set N_k of k new items becomes available. Each new item has a positive weight not greater than b. The aim is to determine a subset U_{n+k}^* of $U_n \cup N_k$ that maximizes the total profit and satisfies the capacity b of the knapsack, knowing the optimal solution U_n^* for Problem K_n. We denote by z_{n+k}^* the maximum profit. An optimal solution of this problem can be easily determined whenever the total weight of the new items is lower than the residual capacity of the knapsack. In fact, in this case, an optimal subset is simply $U_n^* \cup N_k$. Otherwise, an optimal solution is obtained by removing the items of a subset of U_n^* from the knapsack and inserting the items of a subset of $U_n \setminus U_n^* \cup N_k$ into the knapsack such that the resulting profit is maximized and the capacity constraint is satisfied. Obviously, an optimal solution of the Problem RK_{n+k} is equal to an

optimal solution of the Problem K_{n+k}. Our aim is to propose and evaluate the performance of algorithms that, starting from an optimal solution U_n^* of the Problem K_n, determine a heuristic solution of the Problem RK_{n+k}.

In the following, given a set of items T, we denote by $W(T)$ and $P(T)$ the total weight $\sum_{u\in T} w(u)$ and the total profit $\sum_{u\in T} p(u)$ of the items in T, respectively. We also denote by $b(T) = b - W(T)$ the residual capacity after the set of items T has been selected and by $U_n(T)$ the set of items $u \in U_n$ that have a weight $w(u)$ not greater than $b(T)$. Finally, let $z^*(T)$ be the optimum value obtained on set $U_n(T)$ and having a capacity $b(T)$.

2. Computational complexity

We first consider the computational complexity of the simpler problem of deciding whether it is convenient to remove a subset of the items in U_n^* to accommodate a subset of the items in N_k. We show that this problem, referred to as Problem $D - C_{n+k}$, is NP-complete, even in the case in which the set N_k contains a single item, item $n + 1$. Let us define Problem $D - C_{n+1}$.

INSTANCE: A set of items U_n^*, a positive weight $w(u)$ and a positive profit $p(u)$ for each item $u \in U_n^*$, an item $n + 1$ with weight $w(n + 1)$ and profit $p(n + 1)$, $\epsilon > 0$.

QUESTION: Is there a subset $U_n' \subseteq U_n^*$ such that $\sum_{u\in U_n'} p(u) + \epsilon \leq p(n + 1)$ and $\sum_{u\in U_n'} w(u) \geq w(n + 1)$?

Lemma 1. *Problem $D - C_{n+1}$ is NP-complete.*

Proof. We provide a polynomial transformation from the decision version of the knapsack problem, referred to as Problem $D - K_n$. Let us first recall the Problem $D - K_n$, which is well known to be NP-complete (see [5]).

INSTANCE: A capacity B, a finite set I of items, for each item $i \in I$ a positive weight $w(i) \leq B$ and a positive profit $p(i)$, a positive integer K.

QUESTION: Is there a subset $I' \subseteq I$ such that $\sum_{i\in I'} w(i) \leq B$ and such that $\sum_{i\in I'} p(i) \geq K$?

Let us consider an instance of Problem $D - K_n$. We now associate to this instance an instance of Problem $D - C_{n+1}$ built as follows. We associate an item u to each item $i \in I$, so that $U_n^* = I$, and set, for each $u \in U_n^*$, weight $w(u) = p(i)$ and profit $p(u) = w(i)$. The new item $n + 1$ has profit $p(n + 1) = B + \epsilon$ and weight $w(n + 1) = K$. The question has to be answered whether a subset $U_n' \subseteq U_n^*$ exists such that $\sum_{u\in U_n'} p(u) + \epsilon \leq p(n + 1)$ and $\sum_{u\in U_n'} w(u) \geq w(n + 1)$ that is whether $U_n' \subseteq I$ exists such that $\sum_{i\in U_n'} w(i) \leq B$ and $\sum_{i\in U_n'} p(i) \geq K$. This means that the instance of Problem $D - C_{n+1}$ has "yes" answer if and only if the corresponding instance of Problem $D - K_n$ has "yes" answer. □

From this result we obtain the NP-hardness of Problem RK_{n+k} even in the case $k = 1$ by showing that Problem $D-RK_{n+1}$ is NP-complete. Let us define Problem $D - RK_{n+1}$.

INSTANCE: A capacity b, a finite set U_n of items, for each item $u \in U_n$ a positive weight $w(u) \leq b$ and a positive profit $p(u)$, an optimal subset of items $U_n^* \subseteq U_n$, a new item $n + 1$ with weight $w(n + 1) \leq b$ and profit $p(n + 1)$, a positive integer $K > P(U_n^*)$.

QUESTION: Is there a subset $U_n' \subseteq U_n \cup \{n + 1\}$ such that $\sum_{u\in U_n'} p(u) \geq K$ and $\sum_{u\in U_n'} w(u) \leq b$?

Theorem 1. *Problem $D - RK_{n+1}$ is NP-complete.*

Proof. Let us consider an instance of Problem $D - C_{n+1}$, that is a set of items U_n^*, a positive weight $w(u)$ and a positive profit $p(u)$ for each item $u \in U_n^*$, an item $n + 1$ with weight $w(n + 1)$ and profit $p(n + 1)$, $\epsilon > 0$. Let us consider an instance of Problem $D - RK_{n+1}$ where $U_n = U_n^*$ and the items have the same weight and profit, the capacity b is such that $b = W(U_n^*)$ and $K = P(U_n^*) + \epsilon$. The instance of Problem $D - C_{n+1}$ has "yes" as its answer if and only if Problem $D - RK_{n+1}$ has "yes" as its answer. □

3. U_n^*-based approximation algorithms

The first class of heuristic algorithms we study for the solution of Problem RK_{n+k} is based on the idea of determining a solution on the basis of the set $U_n^* \cup N_k$ only, that is on the basis of the information given by an optimal solution of Problem K_n only. In other words, in this class of algorithms, referred to as U_n^*-based approximation algorithms, the items discarded by an optimal solution of Problem K_n cannot be inserted into the reoptimized knapsack. Let z^H denote the value of the objective function obtained by algorithm H. The following result holds.

Theorem 2. *A U_n^*-based approximation algorithm H cannot have a worst-case performance bound greater than* $1/2$.

Proof. Consider the following instance: $|U_n| = 2$, $p(1) = h$, $w(1) = h$, $p(2) = h - 1$, $w(2) = 1$ and $b = h$. The optimal solution of Problem K_n consists in inserting item 1 into the knapsack. Thus, $z_n^* = h$. Now, a new item, item 3, arrives with $p(3) = h - 1$ and $w(3) = h - 1$. Any U_n^*-based approximation algorithm does not insert the new item into the knapsack. In fact, two solutions only have to be evaluated. The first is to keep item 1 in the knapsack; this implies that item 3 cannot be inserted and therefore the corresponding profit is equal to $z_n^* = h$. The second solution is to remove item 1 from the

knapsack. This implies that item 3 only can be inserted with a profit equal to $h-1$, as the algorithm we are considering is U_n^*-based. The best between the two solutions is to keep item 1 in the knapsack, while an optimal solution of Problem RK_{n+1} consists in inserting into the knapsack items 2 and 3 with $z^*_{n+1} = 2h-2$. Thus, $\frac{z^H}{z^*_{n+1}} = \frac{h}{2h-2} \to \frac{1}{2}$ for $h \to \infty$. □

We now propose a U_n^*-based approximation algorithm, referred to as *If-Convenient Heuristic (ICH)*, with tight performance bound 1/2. This algorithm inserts all the items in N_k if $W(N_k) \leq b(U_n^*)$. Otherwise, it finds a subset S of N_k and a subset U' of U_n^* such that S has maximum profit and the constraint $W(S) \leq W(U') + b(U_n^*)$ is satisfied. Then, it replaces the items in U' with the items in S, if convenient, that is if $P(S) > P(U')$.

Let us formally describe the algorithm.

If-Convenient Heuristic (ICH)

1. If $W(N_k) \leq b(U_n^*)$, then insert the items in N_k into the knapsack;
2. Else
   (a) Try all subsets U' of U_n^*; for each U' find a subset S of N_k such that S is a subset of N_k with maximum profit and such that $W(S) \leq W(U') + b(U_n^*)$;
   (b) If $P(S) > P(U')$, then
      i. Remove the items in U' from the knapsack;
      ii. Insert the items in S into the knapsack.

Note that ICH finds the best possible solution that can be obtained while not considering the items discarded in an optimal solution of Problem K_n as it tries all possible subsets of U_n^*. For this reason, the algorithm ICH is not polynomial.

Let z_{ICH} be the profit of the solution generated by ICH. The following result holds.

Theorem 3. *The performance ratio of algorithm ICH is* $\frac{z_{ICH}}{z^*_{n+k}} \geq \frac{1}{2}$ *and the bound is tight.*

Proof. We distinguish the following three cases:

Case 1: $W(N_k) \leq b(U_n^*)$

In this case, $z_{ICH} = z^*_{n+k} = z^*_n + P(N_k)$; therefore, $\frac{z_{ICH}}{z^*_{n+k}} = 1$.

Case 2: $W(N_k) > b(U_n^*)$ and $P(S) > P(U')$:

In this case, $z_{ICH} = z^*_n + P(S) - P(U')$. The optimal profit z^*_{n+k} cannot be greater than the profit of $U_n^* \cup S$, that is $z^*_{n+k} \leq z^*_n + P(S)$. This is due to the fact that S is the subset of N_k with maximum possible profit that satisfies the capacity constraint: A possible subset U' is U_n^* and in this case $W(S)$ must be not greater than the entire capacity b. Since $z_{ICH} \geq z^*_n$ and $z_{ICH} = z^*_n + P(S) - P(U') \geq P(S)$ as $z^*_n \geq P(U')$, then $z^*_{n+k} \leq z^*_n + P(S) \leq 2z_{ICH}$. Therefore,

$$\frac{z_{ICH}}{z^*_{n+k}} \geq \frac{z_{ICH}}{z^*_n + P(S)} \geq \frac{z_{ICH}}{2z_{ICH}} = \frac{1}{2}.$$

Case 3: $W(N_k) > b(U_n^*)$ and $P(S) \leq P(U')$:

In this case, $z_{ICH} = z^*_n$. The maximum profit z^*_{n+k} cannot be greater than the profit of $U_n^* \cup S$, that is $z^*_{n+k} \leq z^*_n + P(S)$. Since $P(S) \leq P(U')$ and $P(U') \leq z^*_n$, then $z^*_{n+k} \leq 2z^*_n$. Therefore,

$$\frac{z_{ICH}}{z^*_{n+k}} \geq \frac{z^*_n}{2z^*_n} = \frac{1}{2}.$$

For the tightness of the bound, we refer to the instance shown in the proof of Theorem 2. □

Note that the result of the previous theorem still holds if the items discarded by an optimal solution of Problem K_n can be inserted after having applied the ICH heuristic.

In conclusion, approximation algorithms based on the information given by an optimal solution of Problem K_n only do not allow us to reduce the worst-case performance bound of 1/2 that can be obtained by applying the classical *Ext-Greedy* algorithm from scratch on the set $U_n \cup N_k$.

4. U_n-based approximation algorithms

We now consider a class of algorithms to solve Problem RK_{n+k}, referred to as U_n-based approximation algorithms, in which the items discarded by an optimal solution of Problem K_n can be used to find a new solution. These algorithms combine the advantages of α-approximation algorithms, with $0 < \alpha < 1$, known for the solution of Problem K_n with the knowledge of an optimal solution of Problem K_n. An example of an α-approximation algorithm is the *Ext-Greedy* algorithm ($\alpha = 1/2$).

We first study the algorithm A^α, the U_n-based approximation algorithm in which a given α-approximation algorithm is used. We derive a bound on the performance of the algorithm A^α, for any available α-approximation algorithm. Then, we show the tightness of the algorithm A^α for some known α-approximation algorithms. The algorithm takes exponential time in k.

The algorithm A^α evaluates, for each nonempty subset S of N_k, the profit that can be obtained by inserting the items of S into the knapsack and then by applying the given α-approximation algorithm on the items in U_n with the residual capacity of the knapsack. We denote by $U_{n+k}^{A^\alpha}$ and $z_{n+k}^{A^\alpha}$ the subset of items selected by this algorithm and the corresponding profit, respectively. Note that the profit of $S = \emptyset$ is the maximum profit of Problem K_n. Thus, $z_{n+k}^{A^\alpha} \geq z_n^*$.

Let $\mathcal{I}_k$ be the set of all subsets S of N_k such that $W(S) \leq b$, and $U_n^\alpha(S)$ be the items selected by applying the given α-approximation algorithm on the set $U_n(S)$ on a knapsack with capacity $b(S)$. The set $S \cup U_n^\alpha(S)$ is the solution of the algorithm A^α when the subset S is chosen and $P(S \cup U_n^\alpha(S))$ the corresponding profit.

The algorithm A^α can be described as follows.

Algorithm A^α

1. For each $S \in \mathcal{I}_k$:
   If $S = \emptyset$ then $U^{A^\alpha}(\emptyset) := U_n^*$ and $z^{A^\alpha}(\emptyset) := z_n^*$
   Else:
   (a) $b(S) := b - W(S)$;
   (b) $U_n(S) := \{u \in U_n : w(u) \leq b(S)\}$;
   (c) Apply the given α-approximation algorithm to the set $U_n(S)$ on a knapsack of capacity $b(S)$ and obtain $U_n^\alpha(S)$
2. $z_{n+k}^{A^\alpha} := \max_{S \in \mathcal{I}_k} P(S \cup U_n^\alpha(S))$; set $U_{n+k}^{A^\alpha}$ equal to the corresponding set of items.

The following result holds.

Theorem 4. *The performance ratio of algorithm A^α is $\frac{z_{n+k}^{A^\alpha}}{z_{n+k}^*} \geq \frac{1}{2-\alpha}$.*

Proof. Suppose that an optimal solution of Problem RK_{n+k} includes the subset of items S^*, $S^* \in \mathcal{I}_k$. Let us distinguish the following two cases.

Case 1: $S^* = \emptyset$

In this case the algorithm obviously finds an optimal solution. Since any optimal solution of Problem RK_{n+k} coincides with an optimal solution of Problem K_n, then $U_{n+k}^{A^\alpha} = U_n^*$.

Case 2: S^* is a nonempty set

In this case

$$z_{n+k}^* = P(S^*) + z^*(S^*),$$

as an optimal solution is given by the items in the set S^* and the items in an optimal solution of the knapsack problem obtained on the set $U_n(S^*)$ with capacity $b - W(S^*)$, with profit $P(S^*)$ and $z^*(S^*)$, respectively. Obviously, by applying the given α-approximation algorithm on the set $U_n(S^*)$ and capacity $b - W(S^*)$ we obtain a solution such that $z^*(S^*) \leq \frac{1}{\alpha} P(U_n^\alpha(S^*))$. Since $z_{n+k}^{A^\alpha} \geq P(S^*) + P(U_n^\alpha(S^*))$, then $z_{n+k}^* = P(S^*) + z^*(S^*) \leq \frac{1}{\alpha} z_{n+k}^{A^\alpha} + (1 - \frac{1}{\alpha}) P(S^*)$ and therefore,

$$z_{n+k}^{A^\alpha} \geq \alpha z_{n+k}^* + (1 - \alpha) P(S^*). \tag{1}$$

Moreover, since $z_{n+k}^{A^\alpha} \geq z^{A^\alpha}(\emptyset) = z_n^*$, then $z_{n+k}^* \leq z_n^* + P(S^*) \leq z_{n+k}^{A^\alpha} + P(S^*)$ and, therefore,

$$P(S^*) \geq z_{n+k}^* - z_{n+k}^{A^\alpha}. \tag{2}$$

Combining (1) with (2), we obtain

$$z_{n+k}^{A^\alpha} \geq \alpha z_{n+k}^* + (1 - \alpha)(z_{n+k}^* - z_{n+k}^{A^\alpha})$$

and, thus, $\frac{z_{n+k}^{A^\alpha}}{z_{n+k}^*} \geq \frac{1}{2-\alpha}$. □

Since the algorithm A^α examines all the subsets of the set of new items N_k with total weight not greater than b, the time complexity is $O((2^k - 1) f(n, 1 - \alpha))$, which is polynomial for any constant k and exponential in k, being $O(f(n, 1 - \alpha))$ the time complexity of the α-approximation algorithm. One may wonder whether an improvement of the performance ratio α of the given approximation algorithm can be achieved with a time complexity that is polynomial in k. The most natural way of constructing such a polynomial time algorithm is to apply the α-approximation algorithm on the set N_k with capacity b and consider a polynomial in k number of subsets of the selected items in A^α instead of all the subsets of N_k. We show that such an algorithm cannot have a worst-case performance better than α.

We consider the algorithm $A^\alpha(\mathcal{I}_k^\alpha)$, identical to A^α with the only exception that the set $\mathcal{I}_k$ of all subsets of N_k with total weight not greater than b is replaced by a subset $\mathcal{I}_k^\alpha$ of it. This subset contains the set of items N_k^α of the solution obtained by applying the α-approximation algorithm on the set N_k with capacity b. In addition, $\mathcal{I}_k^\alpha$ may contain any polynomial in k number of subsets of N_k^α. Let $z^{A^\alpha(\mathcal{I}_k^\alpha)}$ be the profit of the solution obtained by the algorithm $A^\alpha(\mathcal{I}_k^\alpha)$.

Theorem 5. *The algorithm $A^\alpha(\mathcal{I}_k^\alpha)$ cannot have a worst-case performance bound greater than α.*

Proof. Consider the instance in which the total profit $P(U_n)$ of the items in U_n is equal to ϵ, the set N_k and the capacity b of the knapsack are equal to the set of items and the capacity of the worst-case instance of the α-approximation algorithm,

respectively. The profit generated by the algorithm $A^{\alpha}(\mathcal{I}_k^{\alpha})$ is not greater than $P(N_k^{\alpha}) + P(U_n)$, that is $P(N_k^{\alpha}) + \epsilon$. Since the maximum profit z^*_{n+k} is not lower than z^*_k, then

$$\frac{z^{A^{\alpha}(\mathcal{I}_k^{\alpha})}}{z^*_{n+k}} \leq \frac{P(N_k^{\alpha}) + \epsilon}{z^*_k} \to \alpha \quad \text{for } \epsilon \to 0,$$

as $\frac{P(N_k^{\alpha})}{z^*_k} \to \alpha$. □

Therefore, the algorithm $A^{\alpha}(\mathcal{I}_k^{\alpha})$ cannot be better, in the worst case, than the corresponding α-approximation algorithm applied from scratch on the set $U_n \cup N_k$.

Consider the following instance. The set U_n is composed of one item only, say item 1, with profit $p_1 = 1$ and weight $w_1 = \gamma < 1$, and the capacity of the knapsack is $b \geq 1$. The set N_k is composed of two items, say items 2 and 3, with profit $p_2 = 1$ and $p_3 = 1 - \gamma$ and weight $w_2 = b$ and $w_3 = b - \gamma$, respectively. If we consider any algorithm A that selects item 2 in the instance I, then $\frac{z^A}{z^*_{n+k}} \to \frac{1}{2}$ for $\gamma \to 0$. Clearly, algorithm $A^{\alpha}(\mathcal{I}_k^{\alpha})$ would generate this type of solution for any α-approximation algorithm that, when choosing between item 2 and item 3, selects item 2.

We now show the tightness of the algorithm A^{α} for some known α-approximation algorithms.

The algorithm $A^{\frac{1}{2}}$

We first consider the algorithm A^{α} in which the *Ext-Greedy* algorithm is applied on the residual capacity. Since the worst-case performance bound of the *Ext-Greedy* algorithm is 1/2, this algorithm is called algorithm $A^{\frac{1}{2}}$. The following result holds.

Theorem 6. *The performance ratio of algorithm $A^{\frac{1}{2}}$ is $\frac{z^{A^{\frac{1}{2}}}_{n+k}}{z^*_{n+k}} \geq \frac{2}{3}$ and the bound is tight.*

Proof. The proof of the bound is obtained thanks to Theorem 4 when $\alpha = 1/2$. To show that the bound is tight, consider an instance where $b = 2h + 1$, $|U_n| = 3$, $p(1) = p(2) = w(1) = w(2) = h$, $p(3) = 3$ and $w(3) = 2$, $h > 3$. In the optimal solution of Problem K_n the items 1 and 2 are inserted into the knapsack, so that $z^*_n = 2h$. Then, a set N_k of new items arrives with $|N_k| = h$, $p(u) = 1$ and $w(u) = \frac{1}{h}$, $u \in N_k$. None of the solutions considered by the algorithm $A^{\frac{1}{2}}$ will insert both item 1 and item 2 into the knapsack, since the solution given by the *Ext-Greedy* algorithm is to insert item 3, which has the highest efficiency ratio, and either item 1 or 2. It follows that the solution of the algorithm $A^{\frac{1}{2}}$ takes all the items in N_k together with item 1 (or 2) and item 3. Thus, $z^{A^{\frac{1}{2}}}_{n+k} = 2h + 3$. The optimal solution of Problem RK_{n+k} takes all the items in N_k together with items 1 and 2. Thus, $z^*_{n+k} = 3h$. The performance ratio is $\frac{z^{A^{\frac{1}{2}}}_{n+k}}{z^*_{n+k}} \to \frac{2}{3}$ for $h \to \infty$. □

The algorithm $A^{\frac{3}{4}}$

We now consider the algorithm $A^{\frac{3}{4}}$ in which the $G^{\frac{3}{4}}$ algorithm is applied on the residual capacity. The following result holds.

Theorem 7. *The performance ratio of algorithm $A^{\frac{3}{4}}$ is $\frac{z^{A^{\frac{3}{4}}}_{n+k}}{z^*_{n+k}} \geq \frac{4}{5}$ and the bound is tight.*

Proof. The proof of the bound is obtained thanks to Theorem 4 when $\alpha = 3/4$. To show that the bound is tight, consider an instance where $b = 5h$, $|U_n| = 5$, $p(1) = p(2) = p(3) = p(4) = w(1) = w(2) = w(3) = w(4) = h$, $p(5) = 2$ and $w(5) = 1$, $h > 2$. In the optimal solution of Problem K_n all the items are inserted into the knapsack, so $z^*_n = 4h + 2$. Then, a new item arrives, say item 6, with $p(6) = h$ and $w(6) = h$. If the new item is not inserted into the knapsack, the total profit is equal to $z^*_n = 4h + 2$. Otherwise, the algorithm $G^{\frac{3}{4}}$ is applied on the residual capacity equal to $4h$. A pair of items in U_n consists either of two items of profit and weight equal to h or of one item of profit and weight equal to h and item 5. In the first case, two items of profit and weight equal to h are inserted into the knapsack and the *Ext-Greedy* algorithm is applied on the remaining items with a capacity of $2h$. This implies that item 5 and one of the remaining items of profit and weight equal to h are inserted into the knapsack. The total profit of this solution is $4h + 2$. In the second case, one item of profit and weight equal to h and item 5 are inserted into the knapsack. No additional items are inserted into the knapsack by the *Ext-Greedy* algorithm, as the profit of each of the remaining items is greater than the minimum profit of the items already inserted, that is 2. The total profit of this solution is $2h + 2$. It follows that $z^{A^{\frac{3}{4}}}_{n+k} = 4h + 2$. The optimal solution of Problem RK_{n+k} takes all the items with profit and weight equal to h. Thus, $z^*_{n+k} = 5h$. The ratio gives $\frac{z^{A^{\frac{3}{4}}}_{n+k}}{z^*_{n+k}} \to \frac{4}{5}$ for $h \to \infty$. □

4.1. Comparison with PTAS and FPTAS

In Theorem 4 we have shown that the algorithm A^{α} has a worst-case performance bound of $\frac{1}{2-\alpha}$, that is always not worse than the worst-case performance bound α of the approximation algorithm on which it is based (base algorithm). As already

mentioned in the Introduction, very efficient PTAS and FPTAS have been proposed for the solution of the knapsack problem. Such schemes are usually more complex to implement than most of the approximation algorithms. However, it is interesting to compare the running time of the algorithm A^α, when a PTAS or a FPTAS is taken as base algorithm, with the running time of the PTAS or FPTAS run from scratch on the new instance. In this section we will show that there exist classes of instances where the algorithm A^α is more efficient than an *equivalent* PTAS or FPTAS run on the new instance.

Given a $(1-\epsilon)$-approximation algorithm, a PTAS or a FPTAS, we define the algorithm *equivalent* to the algorithm A^α if it guarantees the same worst-case bound $\frac{1}{2-\alpha}$ of the algorithm A^α. This is achieved for $\epsilon = \frac{1-\alpha}{2-\alpha}$. We show that, even if we use the best known PTAS and FPTAS as base algorithms, there exist classes of instances such that the algorithm A^α has a running time lower than an equivalent $(1-\epsilon)$-approximation algorithm.

Let $f(\beta, \epsilon)$ be the running time of the base algorithm on the instances with β items, given that the worst-case performance bound of this algorithm is $1-\epsilon$. Given k new items, the algorithm A^α requires a running time $(2^k-1)f(n, 1-\alpha)$, since $\epsilon = 1-\alpha$. The equivalent $(1-\epsilon)$-approximation algorithm requires a running time $f(n+k, \frac{1-\alpha}{2-\alpha})$, as $\epsilon = \frac{1-\alpha}{2-\alpha}$.

Consider now the PTAS algorithm by Caprara et al. [4] as base algorithm. Its running time is $f(\beta, \epsilon) = \beta^{\frac{1}{\epsilon}-2}$. Given the k new items and $\epsilon = 1-\alpha$, the running time of the algorithm A^α based on this PTAS, referred to as A^α_{PTAS}, is $r^{A^\alpha_{PTAS}} = (2^k-1)n^{\frac{2\alpha-1}{1-\alpha}}$. The running time of the equivalent $(1-\epsilon)$-approximation algorithm is $r^{PTAS} = (n+k)^{\frac{\alpha}{1-\alpha}}$ as $\epsilon = \frac{1-\alpha}{2-\alpha}$. We now show that for $k \leq \log_2 n$, the running time of the latter algorithm is greater than the running time of the algorithm A^α_{PTAS}.

Theorem 8. *For $k \leq \log_2 n$, $r^{PTAS} > r^{A^\alpha_{PTAS}}$.*

Proof. Since for $k = \log_2 n$, $r^{A^\alpha_{PTAS}} = n^{\frac{\alpha}{1-\alpha}} - n^{\frac{2\alpha-1}{1-\alpha}}$, then $r^{A^\alpha_{PTAS}} \leq n^{\frac{\alpha}{1-\alpha}} - n^{\frac{2\alpha-1}{1-\alpha}}$ for $k \leq \log_2 n$. Therefore, for $k \leq \log_2 n$

$$r^{PTAS} = (n+k)^{\frac{\alpha}{1-\alpha}} > n^{\frac{\alpha}{1-\alpha}} > n^{\frac{\alpha}{1-\alpha}} - n^{\frac{2\alpha-1}{1-\alpha}} \geq r^{A^\alpha_{PTAS}}. \quad \square$$

Consider now the FPTAS algorithm by Kellerer and Pferschy [7,8]. Its running time is $f(\beta, \epsilon) = \beta \min\{\log_2 \beta, \log_2 \frac{1}{\epsilon}\} + \frac{1}{\epsilon^2} \log_2 \frac{1}{\epsilon} \min\{\beta, \frac{1}{\epsilon} \log_2 \frac{1}{\epsilon}\}$. Given the k new items and $\epsilon = 1-\alpha$, the running time of the algorithm A^α based on this FPTAS, referred to as A^α_{FPTAS}, is

$$r^{A^\alpha_{FPTAS}} = (2^k-1)\left[n \min\left\{\log_2 n, \log_2 \frac{1}{1-\alpha}\right\} + \frac{1}{(1-\alpha)^2} \log_2 \frac{1}{1-\alpha} \min\left\{n, \frac{1}{1-\alpha} \log_2 \frac{1}{1-\alpha}\right\}\right].$$

The running time of the equivalent $(1-\epsilon)$-approximation algorithm is

$$r^{FPTAS} = (n+k) \min\left\{\log_2(n+k), \log_2 \frac{2-\alpha}{1-\alpha}\right\} + \left(\frac{2-\alpha}{1-\alpha}\right)^2 \log_2 \frac{2-\alpha}{1-\alpha} \min\left\{n+k, \frac{2-\alpha}{1-\alpha} \log_2 \frac{2-\alpha}{1-\alpha}\right\},$$

as $\epsilon = \frac{1-\alpha}{2-\alpha}$.

We now show that for $k = 1$, the running time of the latter algorithm is greater than the running time of the algorithm A^α_{FPTAS}.

Theorem 9. *For $k = 1$, $r^{FPTAS} > r^{A^\alpha_{FPTAS}}$.*

Proof. Since $\frac{2-\alpha}{1-\alpha} > \frac{1}{1-\alpha} > 1$ for $\alpha < 1$, then

$$\left(\frac{2-\alpha}{1-\alpha}\right)^2 \log_2 \frac{2-\alpha}{1-\alpha} \min\left\{n+1, \frac{2-\alpha}{1-\alpha} \log_2 \frac{2-\alpha}{1-\alpha}\right\} > \frac{1}{(1-\alpha)^2} \log_2 \frac{1}{1-\alpha} \min\left\{n, \frac{1}{1-\alpha} \log_2 \frac{1}{1-\alpha}\right\}.$$

Therefore, in order to prove that, for $k = 1$, $r^{FPTAS} > r^{A^\alpha_{FPTAS}}$, it is sufficient to show that

$$(n+1) \min\left\{\log_2(n+1), \log_2 \frac{2-\alpha}{1-\alpha}\right\} \geq n \min\left\{\log_2 n, \log_2 \frac{1}{1-\alpha}\right\}. \tag{3}$$

Let us distinguish the following cases:

Case 1: $\log_2(n+1) \geq \log_2 \frac{2-\alpha}{1-\alpha}$ and $\log_2 n \geq \log_2 \frac{1}{1-\alpha}$. In this case, (3) reduces to $(n+1)\log_2 \frac{2-\alpha}{1-\alpha} \geq n \log_2 \frac{1}{1-\alpha}$, which is obviously satisfied.

Case 2: $\log_2(n+1) \geq \log_2 \frac{2-\alpha}{1-\alpha}$ and $\log_2 n < \log_2 \frac{1}{1-\alpha}$, i.e. $n < \frac{1}{1-\alpha}$. In this case, (3) reduces to $(n+1)\log_2 \frac{2-\alpha}{1-\alpha} \geq n \log_2 n$. Since $\log_2 \frac{2-\alpha}{1-\alpha} > \log_2 \frac{1}{1-\alpha} > \log_2 n$, then $(n+1)\log_2 \frac{2-\alpha}{1-\alpha} > (n+1)\log_2 n > n \log_2 n$. Therefore, (3) is satisfied.

Case 3: $\log_2(n+1) < \log_2 \frac{2-\alpha}{1-\alpha}$ and $\log_2 n \geq \log_2 \frac{1}{1-\alpha}$. In this case, (3) reduces to $(n+1)\log_2(n+1) \geq n \log_2 \frac{1}{1-\alpha}$. Since $\log_2(n+1) > \log_2 n \geq \log_2 \frac{1}{1-\alpha}$, then $(n+1)\log_2(n+1) > (n+1)\log_2 \frac{1}{1-\alpha} > n \log_2 \frac{1}{1-\alpha}$. Therefore, (3) is satisfied.

Case 4: $\log_2(n+1) < \log_2 \frac{2-\alpha}{1-\alpha}$ and $\log_2 n < \log_2 \frac{1}{1-\alpha}$. In this case, (3) reduces to $(n+1)\log_2(n+1) \geq n \log_2 n$, which is obviously satisfied. $\square$

Table 1
Percent increase of the running time of the PTAS with respect to A^{α}_{PTAS}.

n	k	α	%
10	1	0.8	1 364.10
10	1	0.9	2 257.95
10	3	0.8	308.01
10	3	0.9	1 414.93
100	1	0.8	10 306.04
100	1	0.9	10 836.85
100	6	0.8	100.39
100	6	0.9	168.17
1 000	1	0.8	100 300.60
1 000	1	0.9	100 803.61
1 000	9	0.8	102.84
1 000	9	0.9	112.13
10 000	1	0.8	1 000 300.06
10 000	1	0.9	1 000 800.36
10 000	13	0.8	22.72
10 000	13	0.9	23.52
100 000	1	0.8	10 000 300.01
100 000	1	0.9	10 000 800.04
100 000	16	0.8	52.69
100 000	16	0.9	52.81

Table 2
Percent increase of the running time of the FPTAS with respect to A^{α}_{FPTAS}.

n	α	%
10	0.8	74.27
10	0.9	38.37
100	0.8	88.10
100	0.9	43.20
1 000	0.8	34.55
1 000	0.9	35.07
10 000	0.8	14.24
10 000	0.9	14.17
100 000	0.8	11.63
100 000	0.9	5.43

A comparison between the performance of the PTAS by Caprara et al. [4] and the A^{α}_{PTAS} algorithm is proposed in Table 1 for different values of n, k and α. The last column reports the ratio between the running time of the PTAS and the running time of the A^{α}_{PTAS} algorithm. The table shows that A^{α}_{PTAS} outperforms the PTAS especially when $k = 1$ and n is large. A similar comparison is presented in Table 2 with the FPTAS algorithm by Kellerer and Pferschy [7,9]. Here we consider only $k = 1$, as in Theorem 9. The table shows that A^{α}_{FPTAS} outperforms also the FPTAS algorithm.

Other similar results can be obtained for the other known PTAS and FPTAS. Obviously, the class of instances on which the reoptimization algorithm A^{α} is superior to an approximation scheme run from scratch depends on the performance and running time of the scheme. These results show that there always exist situations where it is beneficial to reoptimize.

5. Conclusions

In this paper we have studied the reoptimization of the knapsack problem where a small number k of new items become available after an optimal solution on a large number n of items has been obtained. The problem could be solved by applying an α-approximation algorithm known for the solution of the knapsack problem to solve the instance from scratch on $n + k$ items. We have presented a $\frac{1}{2-\alpha}$-approximation algorithm that makes use of the previously optimal solution and uses a known α-approximation algorithm to solve a subproblem. We have shown that there exist classes of instances on which the running time of the reoptimization algorithm is smaller than the running time of an equivalent PTAS and FPTAS.

The reoptimization problem for the case where a set of items have to be removed is not as interesting as the case we have studied. The reason is that this problem contains instances of the classical knapsack problem, like the one where an optimal set of items contains one item only, say item i, and this is exactly the item that has to be removed.

The study of the reoptimization of other combinatorial problems is of great importance, given the relevance of this setting in practice. Although we believe that the reoptimization of a combinatorial problem is likely to maintain the same

computational complexity of the original problem, it is interesting to study how the knowledge of the previously optimal solution can be beneficial in the reoptimization problem.

Acknowledgements

The authors are grateful to Ulrich Pferschy for his careful reading of a previous version of this paper and for his detailed and valuable comments and suggestions.

The authors also wish to acknowledge the help of two anonymous reviewers who contributed with their suggestions to improve a previous version of this paper.

References

[1] C. Archetti, L. Bertazzi, M.G. Speranza, Reoptimizing the traveling salesman problem, Networks 42 (2003) 154–159.
[2] G. Ausiello, B. Escoffier, J. Monnot, V.Th. Paschos, Reoptimization of minimum and maximum traveling Salesman's tours, in: L. Arge, R. Freivalds (Eds.), Algorithm Theory - SWAT 2006, in: Lecture Notes in Computer Science, vol. 4059, Springer-Verlag, Heidelberg, 2006, pp. 196–207.
[3] H.-J. Böckenhauer, J. Hromkovic, T. Mömke, P. Widmayer, On the hardness of reoptimization, in: SOFSEM 2008: Theory and Practice of Computer Science, in: V. Geffert, et al. (Eds.), Lectures Notes in Computer Science, vol. 4910, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 50–65.
[4] A. Caprara, H. Kellerer, U. Pferschy, D. Pisinger, Approximation algorithms for knapsack problems with cardinality constraints, European Journal of Operational Research 123 (2000) 333–345.
[5] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
[6] O.H. Ibarra, C.E. Kim, Fast approximation algorithms for the knapsack and sum of subset problem, Journal of the ACM 22 (1975) 463–468.
[7] H. Kellerer, U. Pferschy, A new fully polynomial time approximation scheme for the knapsack problem, Journal of Combinatorial Optimization 3 (1999) 59–71.
[8] H. Kellerer, U. Pferschy, Improved dynamic programming in connection with an FPTAS for the knapsack problem, Journal of Combinatorial Optimization 8 (2004) 5–11.
[9] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer–Verlag, Berlin, Heidelberg, 2004.
[10] E.L. Lawler, Fast approximation algorithms for knapsack problems, Mathematics of Operations Research 4 (1979) 339–356.
[11] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons Ltd., Chichester, 1990.
[12] S. Sahni, Approximate algorithms for the 0–1 knapsack problem, Journal of the ACM 22 (1975) 115–124.
[13] M. Schäffter, Scheduling with forbidden sets, Discrete Applied Mathematics 72 (1997) 155–166.